Near parameter free ant colony optimisation

Randall, Marcus

Published in:
Ant Colony Optimization and Swarm Intelligence

DOI:
10.1007/978-3-540-28646-2_37

Near Parameter Free Ant Colony Optimisation

Marcus Randall[1]

Meta-heuristic Search Group
Bond University, QLD 4229, Australia
Ph: +61 7 55953361
Email: mrandall@bond.edu.au

Abstract. Ant colony optimisation, like all other meta-heuristic search processes, requires a set of parameters in order to solve combinatorial problems. These parameters are often tuned by hand by the researcher to a set that seems to work well for the problem under study or a standard set from the literature. However, it is possible to integrate a parameter search process within the running of the meta-heuristic without incurring an undue computational overhead. In this paper, ant colony optimisation is used to evolve suitable parameter values (using its own optimisation processes) while it is solving combinatorial problems. The results reveal for the travelling salesman and quadratic assignment problems that the use of the augmented solver generally performs well against one that uses a standard set of parameter values. This is attributed to the fact that parameter values suitable for the particular problem instance can be automatically derived and varied throughout the search process.

1 Introduction

Meta-heuristic search strategies, including tabu search, simulated annealing, GRASP and ant colony optimisation (ACO), invariably require a set of parameters in order to solve combinatorial optimisation problems. These parameters directly impact on the performance of the solver and as such, researchers and practitioners will often "hand tune" parameter values before the application of the production meta-heuristic or use a set of values that have been found to be traditionally "good" by other researchers (i.e., a standard set).

Relatively little research has been conducted into either the analysis of parameter values or the ways in which they can be automatically derived or tuned by meta-heuristics themselves. In this paper, ant colony optimisation is examined as it is an optimisation framework that has been successfully applied to a range of combinatorial optimisation problems [3, 7]. ACO represents a group of constructive meta-heuristics (often coupled with local search) that use collective intelligence present in insect colonies. The reader is referred to Dorigo and Gambardella [5] and Dorigo and Di Caro [3] for an overview and background of ACO. In this work, ant colony system (ACS) [5] is used as it is a robust and reliable technique.

In regards to ACO studies in which parameters have been analysed, Colorni, Dorigo and Maniezzo [2], Dorigo and Gambardella [4], Dorigo, Maniezzo

and Colorni [6], Maniezzo and Colorni [9] and Shmygelska, Aguirre-Hernández and Hoos [13] have each compared and contrasted various parameter values on particular problems (most notably the travelling salesman problem (TSP) and quadratic assignment problem (QAP)) in order to derive suitable parameter sets.

In terms of automatic parameter adaptation, Ingber [8] developed an implementation known as Adaptive Simulated Annealing in which parameter values are changed in a systematic manner throughout the search process. Pilat and White [10] in their work used concepts from genetic algorithms in order to evolve solution parameters for a constituent ACO technique, ant colony system. This was developed as a "Meta ACS-TSP" that would run standard ACS within a genetic algorithm that evolves solution parameters (a computationally expensive exercise). Using this technique, they were able to suggest alternative good parameter values to Dorigo and Gambardella [5] for ACS and the TSP.

The approach adopted within uses standard mechanisms of ant colony system [5] to modify and to determine appropriate parameter values while problems are being solved. Therefore, it is conceptually simple to integrate this approach into an ant colony implementation (moreso than other search techniques, particularly iterative meta-heuristics). Another advantage is that ant based techniques *learn* appropriate values for particular problem instances (without the researcher/practitioner having to derive these manually). Additionally, it does not add a significant computational overhead to the native algorithm (unlike, for instance, that of Pilat and White [10]). The results show that good quality solutions are achieved for a range of TSP and QAP problem instances. The remainder of the paper is organised as follows. The extensions that allow ACS to evolve its own parameter values (using aspects of the native algorithm) are given in Section 2. Computational experiments, using benchmark TSP and QAP problem instances, are reported in Section 3. Finally, the future directions of this work and conclusions are outlined in Section 4.

2 Evolving ACO Parameter Values

ACS can use the same mechanics for generating solutions to evolve appropriate values for its parameters. Within ACS, the core parameters (apart from the number of ants) are as follows. These are introduced and described in greater detail by Dorigo and Gambardella [5].

- q_0: This parameter determines whether the greedy or probabilistic form of component selection equation is used by an ant at each step of the algorithm. A low value will more likely result in the use of the probabilistic form of the equation (and vice versa).
- ρ: The local pheromone updating factor.
- γ: The global pheromone updating factor.
- β: Is the relative importance placed on the visibility heuristic.

The standard ACS algorithm is augmented at each iteration by allowing each ant to select a value for each parameter before commencing the selection

of the solution components. Thus each ant maintains its own parameter values and in turn uses these to adapt the parameter values.[1] Additionally, a separate pheromone matrix is kept so that the system may learn appropriate parameter values. Selection of a particular value is based exclusively on its pheromone value, as there is no heuristic analogue (such as the distance measure for the TSP) that will determine the quality of a particular parameter value.

Local as well as global pheromone updates are used to adjust the parameter pheromone levels. The γ (global pheromone decay) value of the ant that returns the best quality solution is used in these equations.

Each parameter must be given a suitable range in which its values can lie. It must be noted that the setting of the parameter bounds is not quite the same as setting parameter values. The mechanics of the self-adaptation are designed to quickly identify suitable regions and values for the problem being solved. Due to the nature of the ACS equations, the parameters q_0, ρ, γ are bound between the constant values of 0 and 1. Dorigo and Gambardella [5] have specified that sensible values for β solving the TSP and QAP (as minimisation problems), range between -5 and -1. The initial value of each parameter is chosen as the halfway point in its range. The parameter pheromone matrix is specified as $v(i,j)$, where i represents a particular parameter ($1 \leq i \leq V$), j is the range of values and V is the number of parameters (four in this case). Naturally j is discretised and is bounded between 1 and P. P is a constant the defines the granularity of parameter values. The parameter division, w_i, is chosen using analogues for the solution component selection equations for $v(i,j)$. The actual value of each parameter is then calculated according to Equation 1.

$$p_i = l_i + \frac{w_i}{P}(u_i - l_i) \qquad 1 \leq i \leq V \tag{1}$$

Where:

p_i is the value of the i^{th} parameter,
l_i is the lower bound value of the i^{th} parameter,
w_i is the discretised division chosen for the i^{th} parameter and
u_i is the upper bound value of the i^{th} parameter.

In terms of computational overhead, the use of this scheme adds little burden to the existing ACS search process. As the number of parameters and parameter divisions is fixed, the overall worst case complexity of the ant algorithm remains unaffected.

3 Computational Experiments

The computing platform used to perform the experiments is a 2.6GHz Red Hat Linux (Pentium 4) PC with 512MB of RAM.[2] Each problem instance is run across ten random seeds. Two groups of experiments are performed, one using

[1] It would be also be possible to allow only one ant to modify parameter values.
[2] The experimental programs are coded in the C language and compiled with gcc.

a control strategy (referred to as *control*) and the other using the solver that evolves its own parameter values (referred to as *evolveparam*). The control strategy simply allows the user to manually specify these values. In this case they have been chosen as $\{\beta = -2, \gamma = 0.1, \rho = 0.1, q_0 = 0.9\}$ because these values have been found to be robust by Dorigo and Gambardella [5]. The number of ants, m, and the number of parameter value divisions, P, remain constant at 10 and 20 respectively in these experiments for both *control* and *evolveparam*.

3.1 Algorithm Implementation

The implementations for the TSP and QAP differ slightly due to their respective problem definitions. Each problem, however, requires solutions to be permutations. The objective functions for the TSP and QAP are given by Equations 2 and 3 respectively.

$$\text{Minimise} \sum_{i=1}^{N-1} d(x(i), x(i+1)) + d(x(N), x(1)) \tag{2}$$

Where:

$d(i, j)$ is the distance between cities i and j,
$x(i)$ is the i^{th} city visited and
N is the number of cities.

$$\text{Minimse} \sum_{i=1}^{M} \sum_{j=1}^{M} a(i, j) b(y(i), y(j)) \tag{3}$$

Where:

$a(i, j)$ is the distance between locations i and j,
$b(i, j)$ is the flow between facilities i and j,
$y(i)$ is the facility placed at location i and
M is the number of facilities/locations.

The main differences between the two problems can be characterised by the pheromone representation, visibility heuristic and the transition operators used within the local search phase. The pheromone used for the TSP is given by $\tau(i, j)$ where i and j both represent cities (consistent with Dorigo and Gambardella [5]). For the QAP, $\tau(k, l)$ is used in which k represents the location and l is the facility assigned to the location (in accordance with Maniezzo and Colorni [9]).

For the visibility heuristic, the TSP uses the distance measure between the current city and the potential next city. For the QAP, there are a number of choices for this heuristic, such as the use of a range of approximation functions and not using them at all [14]. The definition used here is given by Equation 4.

$$\eta(w, j) = \sum_{i=1}^{w-1} a(i, w) b(y(i), j) \tag{4}$$

Where:

w is the current location and
j is the the potential facility to assign to $y(w)$.

The local search phase is performed by each ant at every iteration of ACS. The transition operators used for the TSP and QAP are inversion and 2-opt respectively. These operators have been found by Randall and Abramson [11] to provide good performance for these problems. For each operator, the entire neighbourhood is evaluated at each step of the local search phase. The phase is terminated when a better solution cannot be found, guaranteeing a local minimum.

3.2 Problem Instances

Twelve TSP and QAP problem instances are used to test both the effectiveness of the *control* and *evolveparam* strategies. These problems are from TSPLIB [12] and QAPLIB [1] respectively and are given in Table 1.

Table 1. Problem instances used in this study. "Size" for the TSP and QAP is recorded in terms of the number of cities and facilities/locations respectively

Name	Size	Best-Known Cost	Name	Size	Best-Known Cost
hk48	48	11461	nug12	12	578
eil51	51	426	nug15	15	1150
st70	70	675	nug20	20	2570
eil76	76	538	tai25a	25	1167256
kroA100	100	21282	nug30	30	3124
bier127	127	118282	tai35a	35	2422002
d198	198	15780	ste36a	36	9526
ts225	225	126643	tho40	40	240516
gil262	262	2378	sko49	49	23386
pr299	299	48191	tai50a	50	4941410
lin318	318	42029	sko56	56	34458
pcb442	442	50778	sko64	64	48498

3.3 Results

The results are grouped in terms of the problem type. Tables 2 and 3 each show the results of the *control* and *evolveparam* strategies. A particular run of the ACS solver is terminated when a maximum of 3000 iterations of the algorithm has elapsed. This should give the ACS solver sufficient means to adequately explore the search space of these problems. In order to report the results, non-parametric descriptive statistics are used throughout. This is because the distribution of results is highly non-normal. The cost results are reported as the relative percentage deviation (RPD) from the best known solution cost. This is calculated as $\frac{E-F}{F} \times 100$ where E is the result cost and F is the best known cost. The runtime is recorded as the number of CPU seconds required to obtain the best solution within a particular run.

Table 2. The results of the *control* and the *evolveparam* strategies on the TSP instances. Each result is given by a percentage difference (RPD) between the obtained cost and the best known solution. Note that "Min", "Med" and "Max" represent the minimum, median and maximum respectively

Problem	*control*						*evolveparam*					
	Cost			Runtime			Cost			Runtime		
	Min	Med	Max	Min	Med	Max	Min	Med	Max	Min	Med	Max
hk48	0	0.08	0.08	0.04	1.29	16.32	0	0.04	0.08	0.25	3.41	45.02
eil51	0.47	2	2.82	0.08	0.49	40.69	0	0.23	3.52	0.07	9.75	26.42
st70	0.15	1.33	2.07	36.39	43.48	87.56	0	0.89	4.15	8.59	34.9	144.02
eil76	0.19	1.3	2.42	0.08	70.23	114.73	0	0	1.12	2.52	24.58	73.21
kroA100	0	0	0.54	8.67	34.58	192.17	0	0.42	6.32	2.45	195.93	587.38
bier127	0.32	0.72	1.87	58.64	253.21	855.28	0.51	3.63	10.45	5.13	64.14	1248.73
d198	0.16	0.33	0.6	154.53	1723.34	2422.52	0.44	1.69	6.59	10.05	1975.71	4378.37
ts225	0.63	1.15	1.93	513.65	3019.9	5484.59	1.9	3.53	10.84	11.48	611.94	9935.43
gil262	0.63	2.02	2.65	404.07	1674.22	2726.53	0.59	2.78	6.31	121.42	5566.07	19385.71
pr299	0.42	0.92	2.68	10139.87	10794.69	13470.37	0.57	3.36	14.62	23.33	3209.58	19323.87
lin318	1.39	1.92	3	10388.72	14185.36	16090.43	2.19	6.27	15.24	36.11	6020.23	27762.95
pcb442	3.11	3.53	4.39	26903.59	38445.9	57383.08	2.33	12.1	13.13	63.4	199.06	52065.77

Table 3. The results of the *control* and the *evolveparam* strategies on the QAP instances

Problem	*control*						*evolveparam*					
	Cost			Runtime			Cost			Runtime		
	Min	Med	Max	Min	Med	Max	Min	Med	Max	Min	Med	Max
nug12	0	0	0	0	0.02	0.14	0	0	0	0	0	0.04
nug15	0	0	0	0	0.04	0.38	0	0	0	0	0	0.01
nug20	0	0	0	0.02	0.11	2.54	0	0	0	0.02	0.11	7.12
tai25a	0.4	0.63	0.89	2.56	14.45	47.33	0	0.65	1.66	0.09	21.17	50.07
nug30	0	0.07	0.39	1.59	26.79	100.71	0	0.07	0.39	0.27	11.7	32.19
tai35a	0.9	1.34	1.58	84.35	154.21	239.59	0.73	1.41	2.34	1.52	123.51	242.63
ste36a	0	0.39	0.8	33.76	117.47	246.01	0	0	1.6	0.32	63.84	220.34
tho40	0.01	0.16	0.38	16.57	167.27	340.83	0	0.19	1.13	1120.75	1265.13	1275.84
sko49	0.05	0.09	0.25	1.24	6.78	13.38	0.05	0.11	0.35	14.74	113.35	480.7
tai50a	1.87	2.21	2.42	68.91	1060.15	1191.32	1.91	2.18	2.75	1.3	187.79	802.48
sko56	0.02	0.24	0.55	178.62	734.91	1768.55	0	0.18	0.6	42.67	287.06	903.22
sko64	0	0.25	0.6	890.83	2362.08	3553.67	0	0.12	0.58	8.12	298.59	1039.06

The results reveal that overall, *evolveparam* produces very good results compared to *control* on the QAPs and the smaller TSPs. While it sometimes finds better solutions on the large TSPs (such as pcb442) than *control*, frequently its average behaviour is not as good as *control*. Inspection of the runtimes for both strategies shows there is no consistent advantage of using one approach over another (i.e., *evolveparam* does not take considerably longer to run than *control*).

It is interesting to note that the parameter values produced by *evolveparam* often resemble those of Dorigo and Gambardella [5]. The characteristic values of each parameter, in terms of both problems, are summarised as follows:

- q_0: The values were generally between 0.8 and 0.9 for the TSP with the value 0.85 being frequently encountered. However, for the QAPs, lower values, between 0.2 and 0.5 were often evolved.

- β: Generally the range of values was between -2 and -2.5 (with -2.4 being common) for the TSP, while the QAPs tended towards values between -1 and -1.5. Occasionally for both problems, the values would be in the range -4.5 to -5.
- ρ and γ: Values for both parameters were generally in the range of 0.1 to 0.3. Occasionally, larger values (such as 0.85) would be produced for both TSP and the QAP.

A characteristic of *evolveparam* for the larger TSPs is that the values tend to converge to a stable set within approximately a hundred iterations of the algorithm. However, the results for all the QAPs indicated that parameter values were more likely to undergo changes throughout the search process. This correlation between increased performance and the dynamic variation of parameter values warrants further investigation. It would seem appropriate therefore to apply an explicit diversification strategy to the selection of parameter values to counter any premature convergence, particularly for TSPs.

4 Conclusions

Parameter tuning for meta-heuristic search algorithms can be a time consuming and inexact way to find appropriate parameter values to suit various classes of problems. An alternative approach has been explored in this paper in which the algorithmic mechanics of ACS are used to produce and constantly refine suitable values while problems are being solved. The results for the TSP and QAP instances used to test this notion suggest that its performance, in terms of solution costs and runtimes, is comparable to a standard implementation in which values from Dorigo and Gambardella [5] are used. In fact the performance, in terms of objective cost, is often an improvement over the control strategy. This may be attributed to the new solver's ability to tailor parameter values to the problem instance being solved. It is important to note, however, that the parameter values produced by the solver often resemble those of Dorigo and Gambardella [5]. An interesting exercise would be to compare *evolveparam* with a normal ACS that uses optimised parameters for each problem instance to further test its performance.

A concern remains that, at times, the parameter values converge rapidly. This was particularly so for the large TSPs. However, parameter values tended to vary more considerably on the QAPs. This coincided with an increased performance of *evolveparam* strategy over *control*. An extension to this work will be to apply some form of intensification/diversification strategy so that a greater range of values can be explored and/or limiting the pheromone trail values (as done in $\mathcal{MAX} - \mathcal{MIN}$ Ant System [15]). After these extensions are carried out, a more quantitative analysis of the parameter values (and the way in which they change throughout the search process) can be sensibly undertaken. Additionally, the issue of dynamic colony sizes (i.e., varying the value of m, the number of ants) is in the process of being investigated.

References

[1] R. Burkard, S. Karisch, and F. Rendl. QAPLIB - A quadratic assignment problem library. *Journal of Global Optimization*, 10:391–403, 1997.

[2] A. Colorni, M. Dorigo, and V. Maniezzo. An investigation of some properties of an "Ant algorithm". In *Palallel Problem Solving from Nature Conference (PPSN 92)*, pages 509–520, Brussels, Belgium, 1992. Elsevier Publishing.

[3] M. Dorigo and G. Di Caro. The ant colony optimization meta-heuristic. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 11–32. McGraw-Hill, London, 1999.

[4] M. Dorigo and L. Gambardella. A study of some properties of Ant-Q. In *Proceedings of PPSN IV - Fourth International Conference on Parallel Problem Solving From Nature*, Berlin, Germany, 1996. Springer-Verlag.

[5] M. Dorigo and L. Gambardella. Ant Colony System: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, 1997.

[6] M. Dorigo, V. Maniezzo, and A. Colorni. The ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man and Cybernetics - Part B*, 26(1):1–13, 1996.

[7] T. Hendtlass and M. Randall. A survey of ant colony and particle swarm meta-heuristics and their application to discrete optimisation problems. In *Proceedings of the Inaugual Workshop on Artificial Life*, pages 15–25, Adelaide, Australia, 2001.

[8] L. Ingber. Simulated annealing: Practice versus theory. *Computer Modelling*, 18:29–57, 1993.

[9] V. Maniezzo and A. Colorni. The ant system applied to the quadratic assignment problem. *IEEE Transactions on Knowledge and Data Engineering*, 11(5):769–778, 1999.

[10] M. Pilat and T. White. Using genetic algorithms to optimize ACS-TSP. In M. Dorigo, G. Di Caro, and M. Sampels, editors, *Third International Workshop on Ant Algorithms, ANTS 2002*, volume 2463 of *Lecture Notes in Computer Science*, pages 282–287, Brussels, Belgium, 2002. Springer-Verlag.

[11] M. Randall and D. Abramson. A general meta-heuristic solver for combinatorial optimisation problems. *Journal of Computational Optimization and Applications*, 20:185–210, 2001.

[12] G. Reinelt. TSPLIB - A traveling salesman problem library. *ORSA Journal on Computing*, 3:376–384, 1991.

[13] A. Shmygelska, R. Aguirre-Hernandez, and H. Hoos. An ant colony optimization algorithm for the 2D HP protein folding problem. In M. Dorigo, G. Di Caro, and M. Sampels, editors, *Third International Workshop on Ant Algorithms, ANTS 2002*, volume 2463 of *Lecture Notes in Computer Science*, pages 40–52, Brussels, Belgium, 2002. Springer-Verlag.

[14] T. Stützle and M. Dorigo. ACO algorithms for the quadratic assignment problem. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 33–50. McGraw-Hill, London, 1999.

[15] T. Stützle and H. Hoos. The $\mathcal{MAX} - \mathcal{MIN}$ Ant System and local search for combinatorial optimization problems. In S. Voss, S. Martello, I. Osman, and C. Roucairol, editors, *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, pages 313–329. Kluwer Academic Publishers, Boston, MA, 1998.